CEAP

California Enterprise Architecture Program

SOA *White Paper*

Web Services

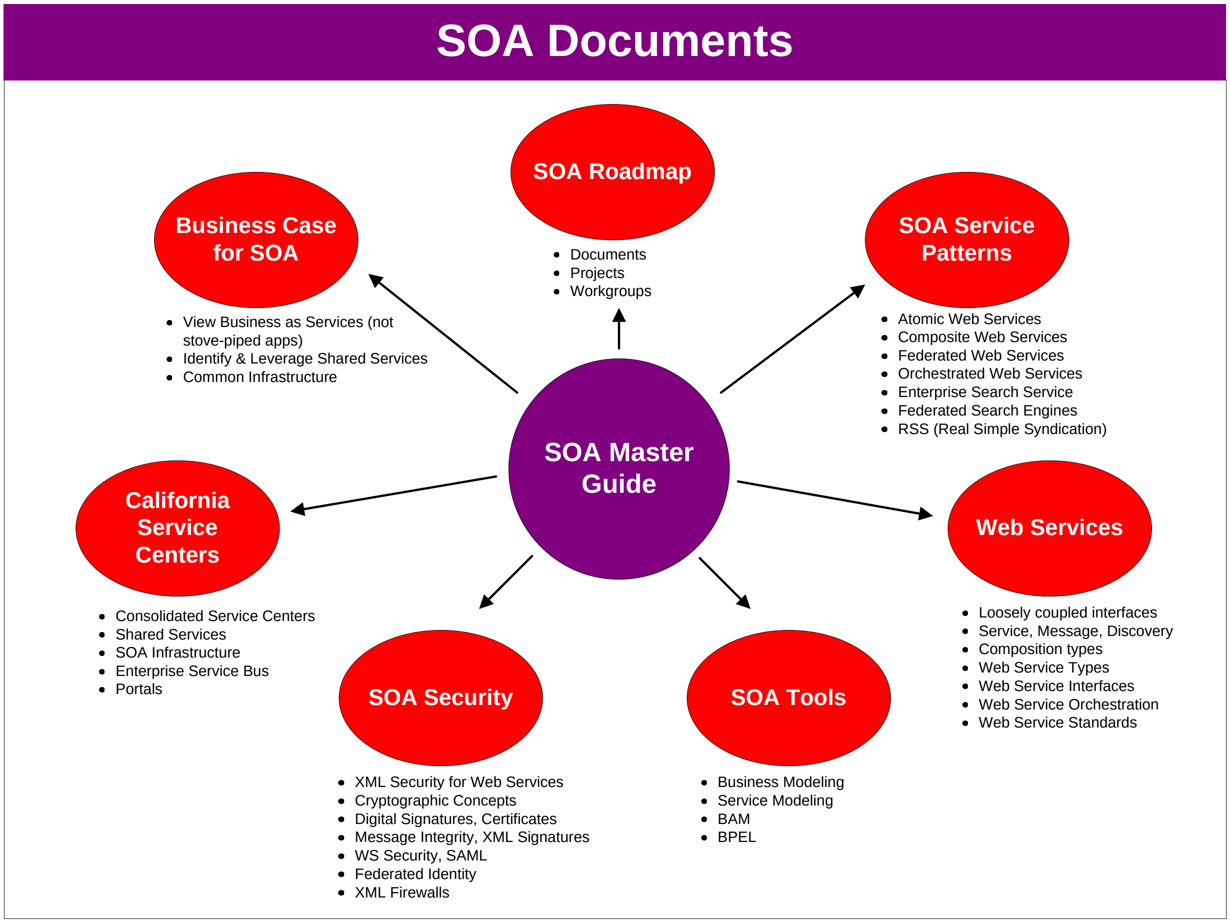
Revision History

06/22/2006	Original Draft

Table of Contents

SOA Documents

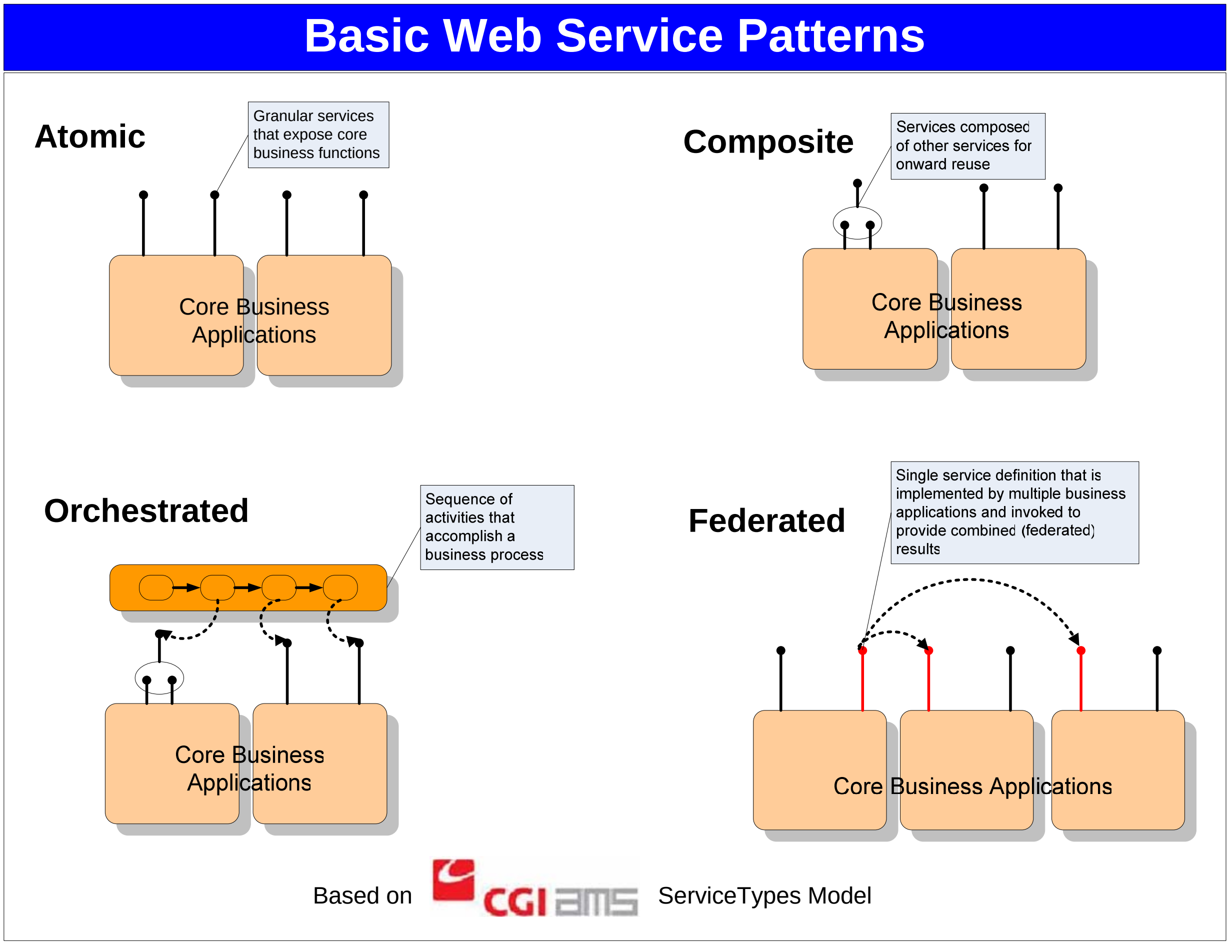
The service oriented architecture advocated by the California Enterprise Architecture Program is organized into a set of interrelated documents. A master guide serves as the "jumping off point" and describes in an overview fashion the key parts of SOA.



There are six white papers planned to address in depth details of SOA. This whitepaper is *Web Services.*

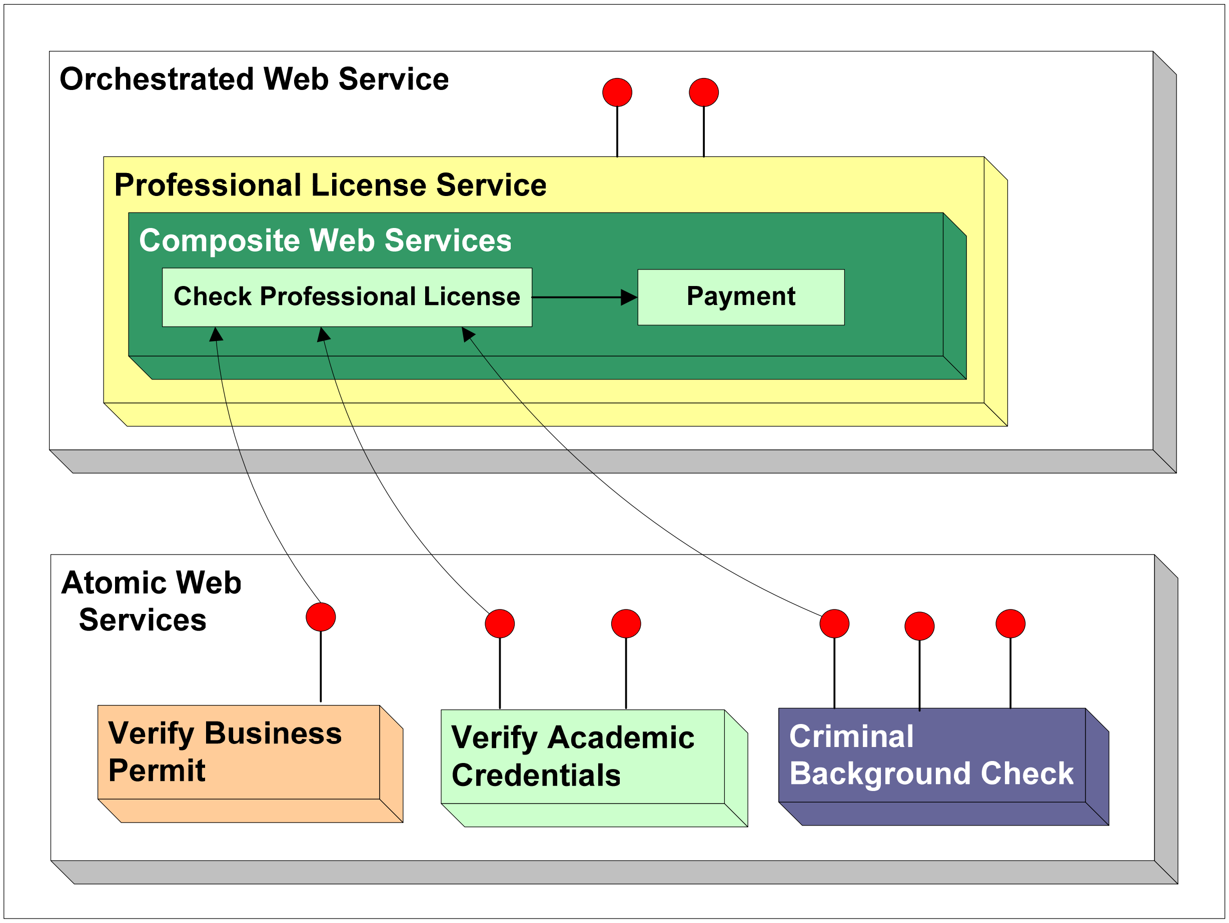
QuickView – Web Services

Web services are a great example of how IT can better support business goals. One way is to consolidate like business functionality which currently is spread across many applications into one common shared service. This saves IT costs and makes it much easier to implement a change to a business process. If the shared services are architected correctly, they can be reused and repurposed to fit many business scenarios resulting in a much richer user experience. Users can then do more of the work instead of a state employee doing it on their behalf which should result in fewer errors, faster recording, and lower costs. The state employees could then be redeployed into more productive jobs that would result in more value delivered to citizens.


Basic Web Service Patterns
Atomic
Granular services that expose core business functions
Core Business Applications
Composite
Services composed of other services for onward reuse
Core Business Applications
Orchestrated
Sequence of activities that accomplish a business process
Core Business Applications
Federated
Single service definition that is implemented by multiple business applications and invoked to provide combined (federated) results
Core Business Applications
Based on CGI ams ServiceTypes Model

Web Services are a key component of a service-oriented architecture. Shared business services are implemented as *atomic*, *composite*, *federated*, or *orchestrated* web services. A composite web service is made up of two or more atomic web services resulting in a new web service with a single interface representing both atomic services. That is, they become less granular with a broader purposed interface. Orchestrated web services are atomic, composite, or federated web services linked together to fulfill some or all of the steps within a given business process. From a business perspective, this means changes to business functionality can be implemented more quickly.

For example, atomic web services might be Verify Business Permits, Verify Academic Credentials, and Criminal Background Check. These three atomic web services might be combined ("composed") into a new Check Professional License web service. This composite service might then be linked with other web services in a Professional License Service which is an orchestration of multiple web services executed in a particular order which together meet all the requirements of the Apply for Professional License business process.


Orchestrated Web Service
Professional License Service
Composite Web Services
Check Professional License
Payment
Atomic Web Services
Verify Business Permit
Verify Academic Credentials
Criminal Background Check

Web services are implemented via XML document ("messaging") interfaces. This is why they are considered highly interoperable. They are language and platform agnostic, meaning it doesn't matter whether they are written in .NET or Java or run on Windows, Unix, or the Mainframe. They simply must "speak" XML and support web service standards. From a business perspective, this means lower IT costs because the interoperability issues are far less complex and standards-based.

A lot of acronyms are used throughout this document. Please refer to the Web Service Standards section for definitions.

Web Services

In order for SOA to be widely adopted, a practical standardized implementation mechanism must exist. Most Web Services are defined in WSDL (XML) and use standard protocols to communicate (SOAP/HTTP). So, using Web Services appears to be the practical solution to implementing an SOA. Alternatively, one could use HTTP-Get or HTTP-Post without SOAP however the data is limited to named-value pairs. SOAP is much more flexible and can handle complex types such as datasets, classes, and other objects.

Web Services are comprised of many components (see Web Service Standards in this document). Here are a few of the most common. Each of these plays an essential role in SOA.

Service

A service in SOA is an application function packaged as a reusable component for use in a business process. It either provides information or facilitates a change to business data from one valid and consistent state to another. The process used to implement a particular service does not matter, as long as it responds to your commands and offers the quality of service you require.

Through defined communication protocols, services can be invoked that stress interoperability and location transparency. A service has the appearance of a software component, in that it looks like a self-contained function from the service requester's perspective. However, the service implementation may actually involve many steps executed on different computers within one enterprise or on computers owned by a number of business partners. A service might or might not be a component in the sense of encapsulated software. Like a class object, the requester application is capable of treating the service as one.

Web services are based on invocation using SOAP messages which are described using WSDL over a standard protocol such as HTTP. Use of Web services is a best practice when communicating with external business partners.

For example, one might query a web services repository to find a list of services that provide doctor or real estate licensing. In this case, it might return Professional License Service, Medical Doctor License Service, Real Estate License Service, Medical Doctor License Verification Service, Medical Doctor Education Verification Service, etc.

“Individual” services, such as Medical Doctor License Verification Service and Medical Doctor Education Verification Service are built with a granular set of functionality. They can be combined into “composite” services such as Medical Doctor License Service which is coarse-grained. Or, they can be wrappered to handle requirements that are not included in the service interfaces. (see Web Service Types)

Message

Service providers and consumers communicate via messages. Services expose an interface which defines the behavior of the service and the messages they accept and return. According to Gartner, “a service interface should specify three facets: Identifiers, Formats, and Protocols. This is a network concept that is used to ensure the loose coupling of network components across the global Internet.

Identifiers are the names or "addresses" of resources, eg URLs. Formats are the message structures, or in the case of XML, the document structures. And protocols are the rules of interaction between the consumer and provider, eg the message exchange pattern." Because the interface is platform and language independent, the technology used to define messages must also be agnostic to any specific platform/language. Therefore, messages are constructed using XML documents that conform to XML schema.   XML provides all of the functionality, granularity, and scalability required by messages. That is, for consumers and providers to effectively communicate, they need a non-restrictive type of system to clearly define messages; XML provides this.

Because consumers and providers communicate via messages, the structure and design of messages should not be taken lightly. Messages need to be implemented using a technology that supports the scalability requirements of services. While XML interfaces are designed to be extensible, having to redesign entire interfaces due to unanticipated changes will break existing consumers and providers which can prove to be costly.

Dynamic Discovery

Dynamic discovery is an important piece of SOA. At a high level, one searches the registry, gets a URL, and downloads the WSDL file. The directory service is an intermediary between providers and consumers. Providers register with the directory service and consumers query the directory service to find service providers. Most directory services organize services based on criteria and categorize them. Consumers can then use the directory services' search capabilities to find providers. Embedding a directory service within SOA accomplishes the following:

1. Scalability of services; you can add services incrementally.
2. Decouples consumers from providers.
3. Allows for hot updates of services.
4. Provides a look-up service for consumers.
5. Allows consumers to choose between providers at runtime rather than hard-coding a single provider.

Web Service Analogy

Although the concepts behind SOA were established long before web services came along, web services play a major role in a SOA. This is because web services are built on top of well-known, platform-independent protocols. These protocols include HTTP, XML, UDDI, WSDL, and SOAP. It is the combination of these protocols that make web services so attractive. Moreover, it is these protocols that fulfill the key requirements of a SOA. That is, a SOA requires that a service be dynamically discovered and invoked. This requirement is fulfilled by UDDI, WSDL, and SOAP. SOA requires that a service have a platform-independent interface. This requirement is fulfilled by XML. SOA stresses interoperability. This requirement is fulfilled by HTTP. This is why web services lie at the heart of SOA.

Acronyms	**Practical Examples**
UDDI	Phone Book
WSDL	Contract
SOAP	Envelope
HTTP, SMTP, FTP	Mail person
Programing (Java, Servlet, ASP.NET, C#)	Speech
Schema	Vocabulary

XML	Alphabet

Simplifying Web Service Terms

The basic steps for locating and calling a web service are illustrated in the following drawing (from Patricia Seybold Group).

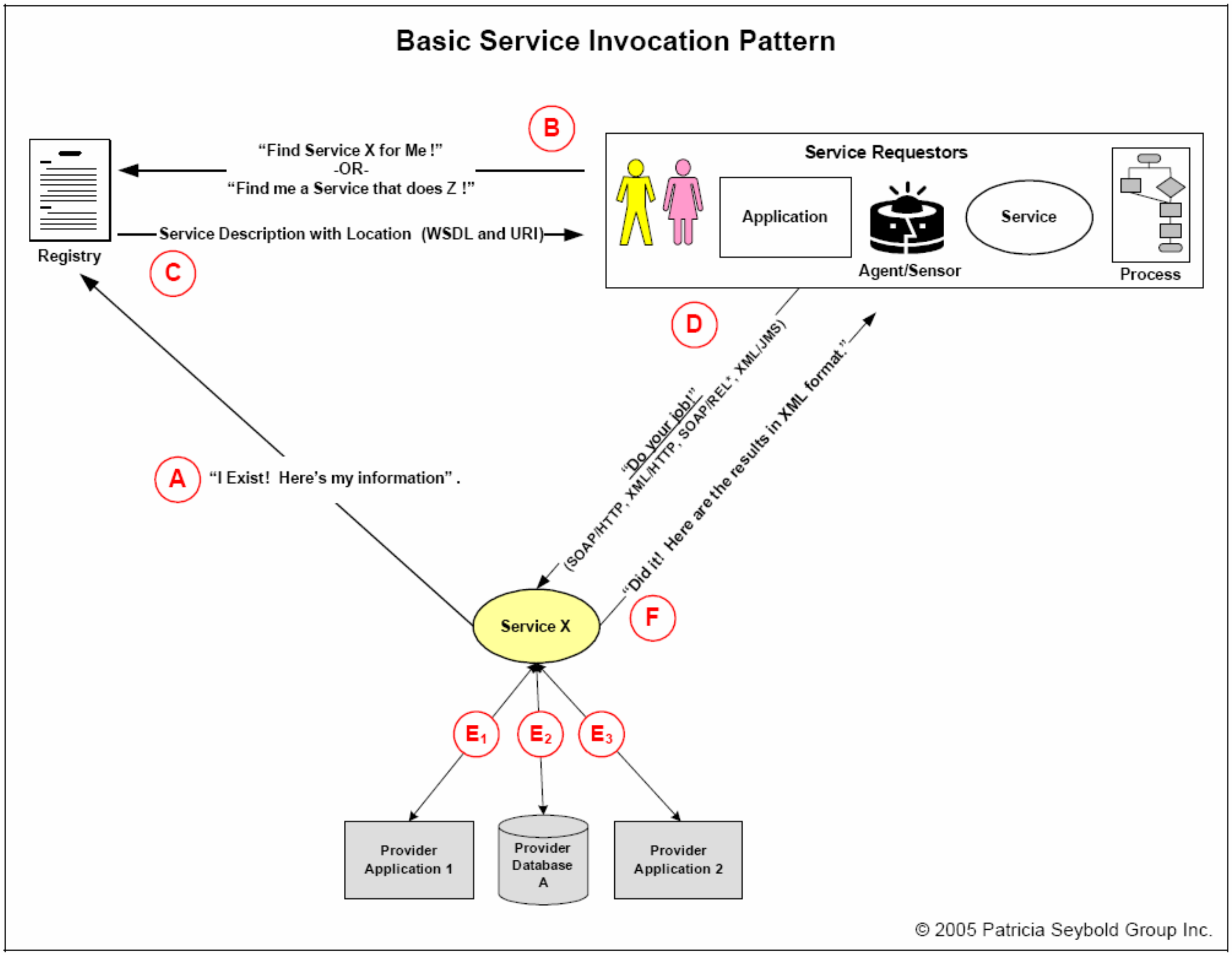
A – Providers register their web services with a common registry, based on a standard such as UDDI. This includes the location and a detailed description of the service in the form of a WSDL XML document.

B – An application (or a different web service) invokes the service from provider A.

C – A SOAP message is sent to the end point of the provider service.

E – The service processes the request based on its internal functionality, which is hidden from the external user.

D – The service returns the results in XML format to the requesting application.


Basic Service Invocation Pattern
B
"Find Service X for Me !"
-OR-
"Find me a Service that does Z !"
Service Description with Location (WSDL and URI)
Registry
C
Service Requestors
Application
Agent/Sensor
Service
Process
D
"Do your job!"
(SOAP/HTTP, XML/HTTP, SOAP/REL*, XML/JMS)
A
"I Exist! Here's my information".
"Did it! Here are the results in XML format."
F
Service X
E1
E2
E3
Provider Application 1
Provider Database A
Provider Application 2
© 2005 Patricia Seybold Group Inc.

Web Service Composition

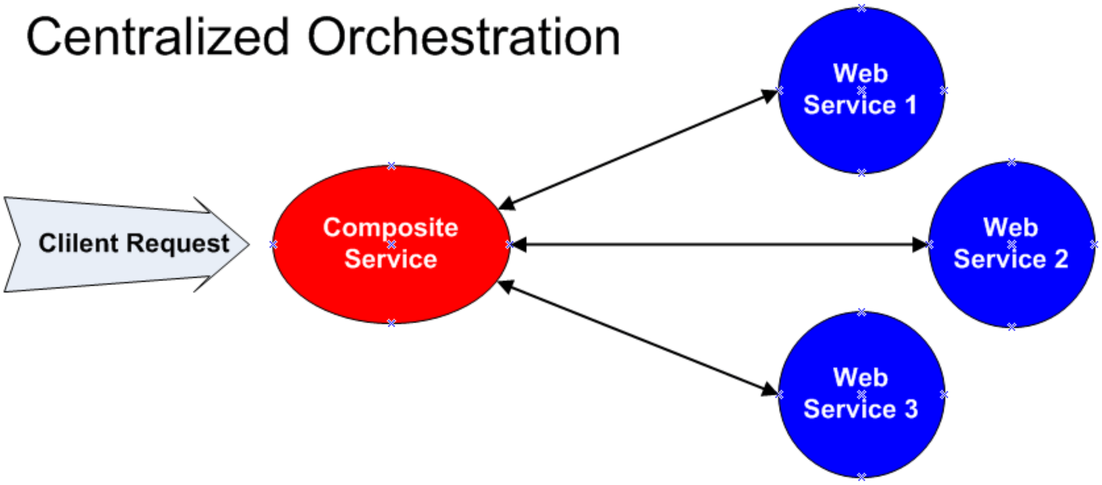
Atomic Services

Web services encapsulate information, software or other resources, and make them available over the network via standard interfaces and protocols. Web service architectures are based on the notion of building a library of specific *atomic* services. Generally, the more the more simple and generic the functionality the better the chances are that the service can be used in multiple applications. For example, Address Verification Service, Credit Card Payment Service, and Education Verification Service are candidates for atomic web services.

Composite Services

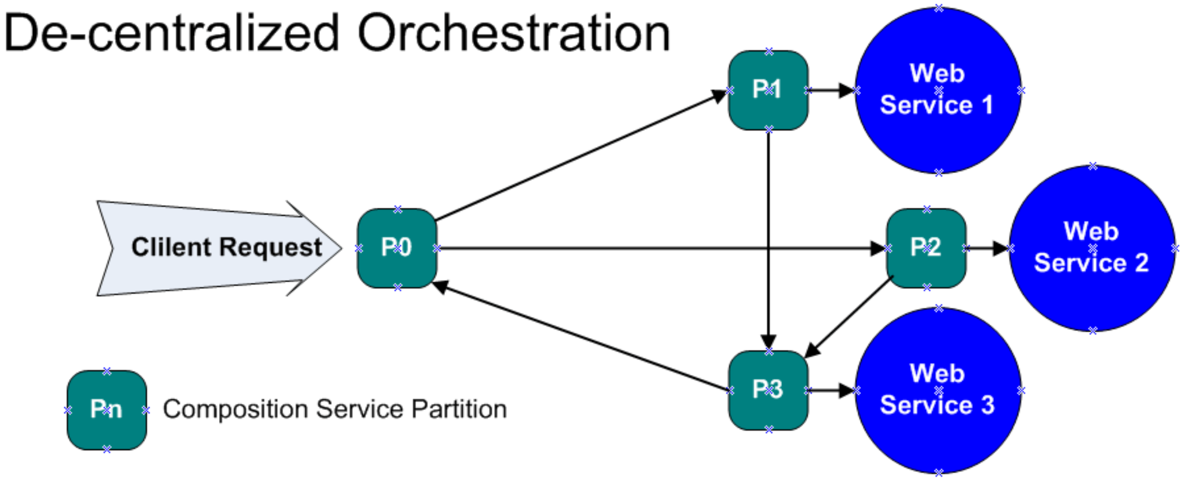
A second powerful notion focuses on aggregating atomic web services into larger, less granular services. Complex web services may be created by aggregating the functionality provided by simpler ones. This is referred to as service composition and the aggregated web service becomes a *composite* web service. For example, Dentist License Verification Service and Dentist Education Verification Service might be rolled into Dentist Qualifications Service.

Composite web services may be developed using a specification language such as BPEL and executed by a workflow engine. Typically, a composite web service specification is executed by a single coordinator node. It receives the client requests, makes the required data transformations and invokes the component web services as per the specification. This mode of execution is known as *centralized orchestration.*



In *decentralized orchestration* of composite web services, there are multiple engines, each executing a composite web service specification (a portion of the original composite web service specification but complete in itself) at distributed locations. The engines communicate directly with each other (rather than through a central coordinator) to transfer data and control when necessary in an asynchronous manner.

Decentralized orchestration is often referred to as *choreography*.



Web Service Types

(This section is an excerpt from: XML.COM http://www.xml.com/pub/a/ws/2003/09/30/soa.html)

There are two main styles of Web services: SOAP web services and REST web services.

SOAP Web services

A SOAP web service introduces the following constraints:

1. Except for binary data attachment, messages must be carried by SOAP.
2. The description of a service must be in WSDL.

A SOAP web service is the most common and marketed form of web service in the industry. Some people simply collapse "web service" into SOAP and WSDL services. SOAP provides "a message construct that can be exchanged over a variety of underlying protocols" according to the SOAP 1.2 Primer. In other words, SOAP acts like an envelope that carries its contents. One advantage of SOAP is that it allows rich message exchange patterns ranging from traditional request-and-response to broadcasting and sophisticated message correlations. There are two flavors of SOAP web services, SOAP RPC and document-centric SOAP web service. SOAP RPC web services are not SOA; document-centric SOAP web services are SOA.

A SOAP RPC web service breaks the second constraint required by an SOA. A SOAP RPC Web service encodes RPC (remote procedure calls) in SOAP messages. In other words, SOAP RPC "tunnels" new application-specific RPC interfaces though an underlying generic interface. Effectively, it prescribes both system behaviors and application semantics. Because system behaviors are very difficult to prescribe in a distributed environment, applications created with SOAP RPC are not interoperable by nature. Many real life implementations have confirmed this.

Faced with this difficulty, both WS-I basic profile and SOAP 1.2 have made the support of RPC optional. RPC also tends to be instructive rather than descriptive, which is against the spirit of SOA. Ironically, SOAP was originally designed just for RPC. It won't be long before someone claims that "SOAP" actually stands for "SOA Protocol".

REST Web Services

The term REST was first introduced by Roy Fielding to describe the web architecture. A REST web service is an SOA based on the concept of "resource". A resource is anything that has a URI. A resource may have zero or more representations. Usually, people say that a resource does not exist if no representation is available for that resource. A REST web service requires the following additional constraints:

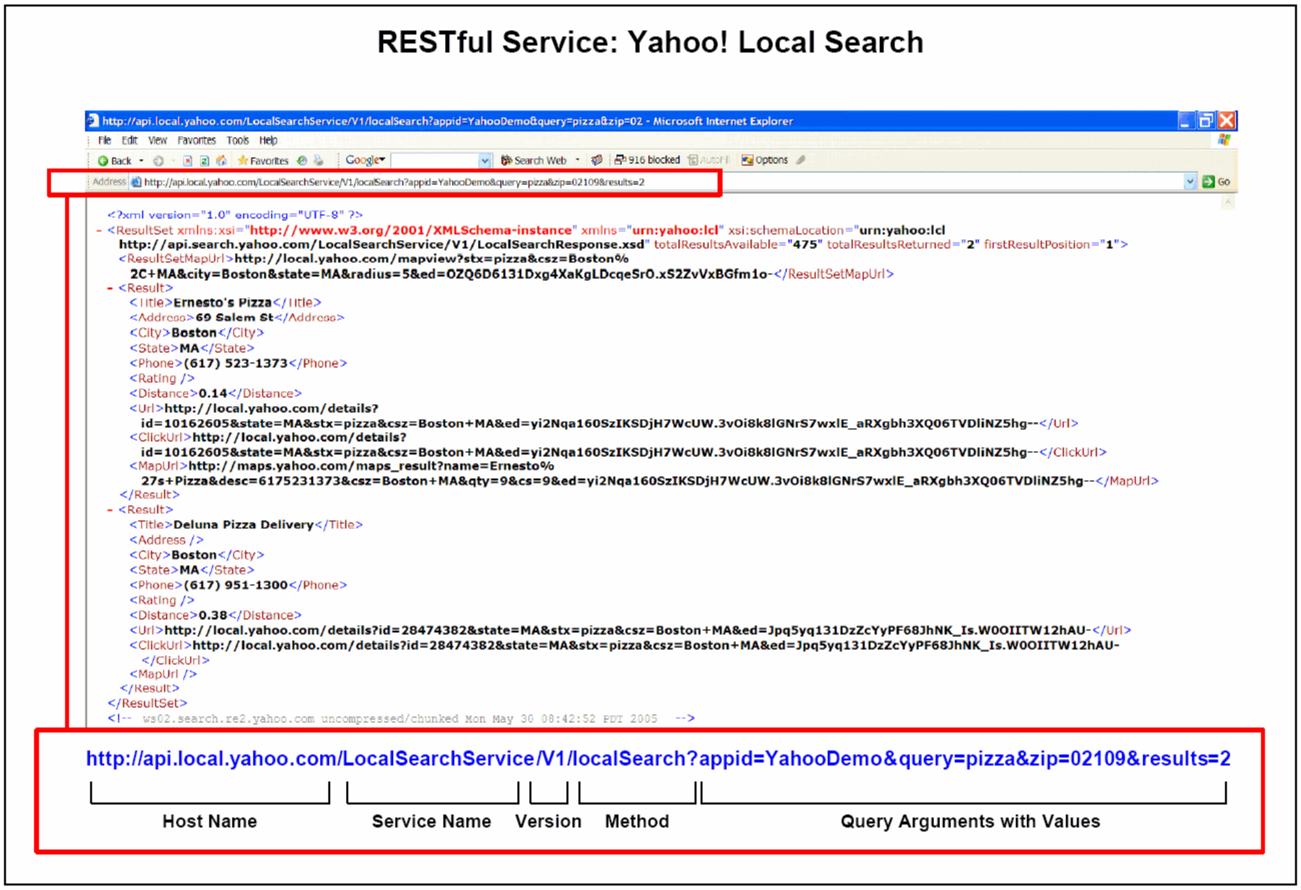
1. Interfaces are limited to HTTP. The following semantics are defined:

- HTTP GET is used for obtaining a representation of a resource. A consumer uses it to retrieve a representation from a URI. Services provided through this interface must not incur any obligation from consumers.
- HTTP DELETE is used for removing representations of a resource.
- HTTP POST is used for updating or creating the representations of a resource.
- HTTP PUT is used for creating representations of a resource.

2. Most messages are in XML, confined by a schema written in a schema language such as XML Schema from W3C or RELAX NG.
3. Simple messages can be encoded with URL encoding.
4. Service and service providers must be resources while a consumer can be a resource.

REST web services require little infrastructure support apart from standard HTTP and XML processing technologies, which are now well supported by most programming languages and platforms. REST web services are simple and effective because HTTP is the most widely available interface, and it is good enough for most applications. In many cases, the simplicity of HTTP simply outweighs the complexity of introducing an additional transport layer. *– End of XML.COM referenced material.*

One example of a RESTful Web Service is the Yahoo! Local Search service (with results) shown below. It shows the popular practice of encoding all of the information required to invoke the service in the URL. As you can see, the entire interaction took place within the browser. Entering the service's URL in the browser address line returned an XML document with the name and location of pizza shops near our office. The simplicity offered by REST makes it attractive in situations where only simple name-value pairs are required.



RESTful Service: Yahoo! Local Search

http://api.local.yahoo.com/LocalSearchService/V1/localSearch?appid=YahooDemo&query=pizza&zip=02 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Favorites
Google
Search Web
916 blocked
AutoFill
Options
Address http://api.local.yahoo.com/LocalSearchService/V1/localSearch?appid=YahooDemo&query=pizza&zip=02109&results=2
Go

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <ResultSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="urn:yahoo:lcl" xsi:schemaLocation="urn:yahoo:lcl
    http://api.search.yahoo.com/LocalSearchService/V1/LocalSearchResponse.xsd" totalResultsAvailable="475" totalResultsReturned="2" firstResultPosition="1">
    <ResultSetMapUrl>http://local.yahoo.com/mapview?stx=pizza&csz=Boston%
      2C+MA&city=Boston&state=MA&radius=5&ed=0ZQ6D6131Dxg4XaKgLDcqeSr0.xS2ZvVxBGfm1o-</ResultSetMapUrl>
  - <Result>
      <Title>Ernesto's Pizza</Title>
      <Address>69 Salem St</Address>
      <City>Boston</City>
      <State>MA</State>
      <Phone>(617) 523-1373</Phone>
      <Rating />
      <Distance>0.14</Distance>
      <Url>http://local.yahoo.com/details?
        id=10162605&state=MA&stx=pizza&csz=Boston+MA&ed=yi2Nqa160SzIKSDjH7WcUW.3vOi8k8lGNrS7wxlE_aRXgbh3XQ06TVDliNZ5hg--</Url>
      <ClickUrl>http://local.yahoo.com/details?
        id=10162605&state=MA&stx=pizza&csz=Boston+MA&ed=yi2Nqa160SzIKSDjH7WcUW.3vOi8k8lGNrS7wxlE_aRXgbh3XQ06TVDliNZ5hg--</ClickUrl>
      <MapUrl>http://maps.yahoo.com/maps_result?name=Ernesto%
        27s+Pizza&desc=6175231373&csz=Boston+MA&qty=9&cs=9&ed=yi2Nqa160SzIKSDjH7WcUW.3vOi8k8lGNrS7wxlE_aRXgbh3XQ06TVDliNZ5hg--</MapUrl>
    </Result>
  - <Result>
      <Title>Deluna Pizza Delivery</Title>
      <Address />
      <City>Boston</City>
      <State>MA</State>
      <Phone>(617) 951-1300</Phone>
      <Rating />
      <Distance>0.38</Distance>
      <Url>http://local.yahoo.com/details?id=28474382&state=MA&stx=pizza&csz=Boston+MA&ed=Jpq5yq131DzZcYyPF68JhNK_Is.W0OIITW12hAU-</Url>
      <ClickUrl>http://local.yahoo.com/details?id=28474382&state=MA&stx=pizza&csz=Boston+MA&ed=Jpq5yq131DzZcYyPF68JhNK_Is.W0OIITW12hAU-
        </ClickUrl>
      <MapUrl />
    </Result>
  </ResultSet>
  <!-- ws02.search.re2.yahoo.com uncompressed/chunked Mon May 30 08:42:52 PDT 2005 -->
```

http://api.local.yahoo.com/LocalSearchService/V1/localSearch?appid=YahooDemo&query=pizza&zip=02109&results=2

In contrast, SOAP would be used where more complex structures are required such as classes and datasets.

Web Services with Presentation Logic

From a standards perspective, there is work underway on two fronts when it comes to coupling presentation with services.

Web Services for Remote Portlets (WSRP)

Most web services are designed to handle business logic and data manipulation. However, there is an emerging standard, Web Services for Remote Portlets (WSRP) that is intended to accommodate presentation logic. So, utilizing this standard one could use a web service to generate web page content.

RSS (Real Simple Syndication) is an example of REST-style web services with presentation markup in the results. Some people state the REST style of xml presentation has achieved more success on the Web than WSRP.

Asynchronous JavaScript and XML (AJAX)

Ajax, shorthand for Asynchronous **JavaScript** and **XML**, is a **Web** development technique for creating interactive **web applications**. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user makes a change. This is meant to increase the web page's interactivity, speed, and **usability**. -- *Wikipedia*

A key part of the AJAX is the XMLHttpRequest object. This object, first implemented by Microsoft as an ActiveX object but now also available as a native object within both Mozilla and Apple's Safari browser, enables JavaScript to make HTTP requests to a remote server without the need to reload the page. In essence, HTTP requests can be made and responses received, completely in the background and without the user experiencing any visual interruptions.

So, AJAX enabled web applications can make calls to web services to retrieve data in XML format and update just a section of the web page.

Here are some good links to more information on AJAX:
http://en.wikipedia.org/wiki/AJAX
http://adaptivepath.com/publications/essays/archives/000385.php
http://www.xml.com/pub/a/2005/02/09/xml-http-request.html

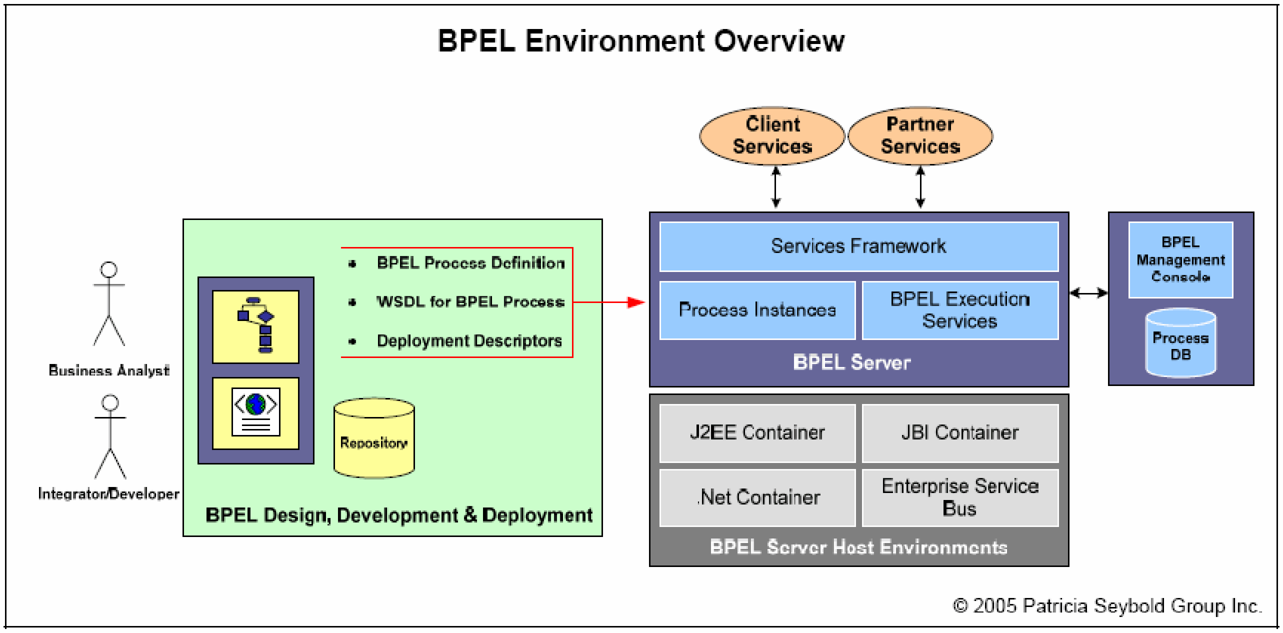
Web Service Patterns

There are four basic web service patterns: *atomic*, *composite*, *federated*, and *orchestrated*. Additionally, three topic specific patterns have been defined. Two of them suggest ways to handle enterprise search and the third illustrates using the industry standard RSS (Real Simple Syndication) for publishing practically any type of public information.

See SOA Service Patterns White Paper for details on each of the above seven patterns.

Orchestration Example

An important specification for enterprise integration and service-oriented architecture is business process execution language (BPEL). In BPEL, a business process is a large-grained stateful service, which executes steps to complete a business goal. That goal can be the completion of a business transaction, or fulfilling the job of a service. The steps in the BPEL process execute activities (represented by BPEL language elements) to accomplish work. Those activities are centered on invoking partner services to perform tasks (their job) and return results back to the process. The aggregate work, the collaboration of all the services, is a service orchestration.



It is important to understand the best uses (and limitations) of BPEL. BPEL offers a nice model to abstract orchestration logic from the participating services, and configuration using BPEL over (hard core) coding of service inter-actions is enticing. However, there is processing overhead and infrastructure expense, so BPEL might not be the best choice for simple orchestrations. As a rule of thumb, a simple orchestration is comprised of two to five services and has static interaction patterns.

As a language to develop processes, BPEL is good at executing a series of activities, which occur over time, and interact with internal and external services. These processes may represent IT scenarios, such as integration, or business scenarios, such as information exchange, or flows of work.

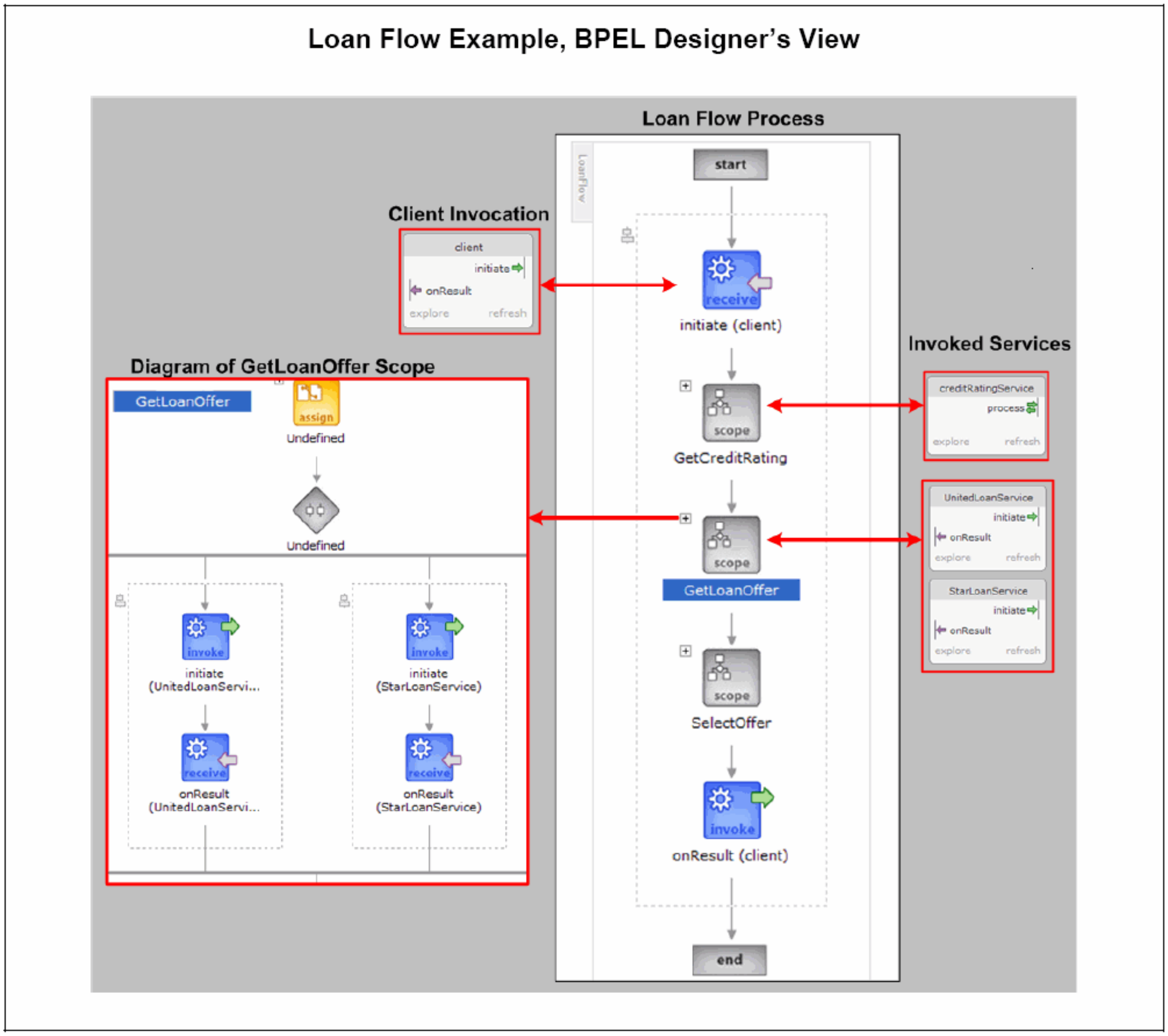
As for limitations, BPEL does not account for humans in a process, so BPEL doesn't provide workflow - there are no concepts for roles, tasks and inboxes. In addition, BPEL does not support really complex business processes, which evolve during their execution, branching out to incorporate new parties and activities. Lastly, BPEL does not have native support for business activity monitoring (BAM). There isn't a data model for measurement and monitoring.

Here is a modified simple Loan Flow from Oracle's BPEL demonstration samples. The business flow is as follows:

1. A customer requests a loan quote
2. The loan flow process starts and performs a customer credit check

The following picture shows the process from a BPEL designer's view. In the center of the diagram is the Loan Flow process. To the right of the loan flow are the invoked (partner) services. At the top left is the client invocation <receive>.

Using a BPEL design tool, you could click on any of the elements to expand the information and make modifications. Shown is an expansion of the GetLoanOffer scope. Here are the invocations of the lending institutions' services. The structure for the invocations is <flow>, indicating concurrent requests. When both institutions have replied, the GetLoanOffer scope completes, and processing continues to the SelectOffer scope.



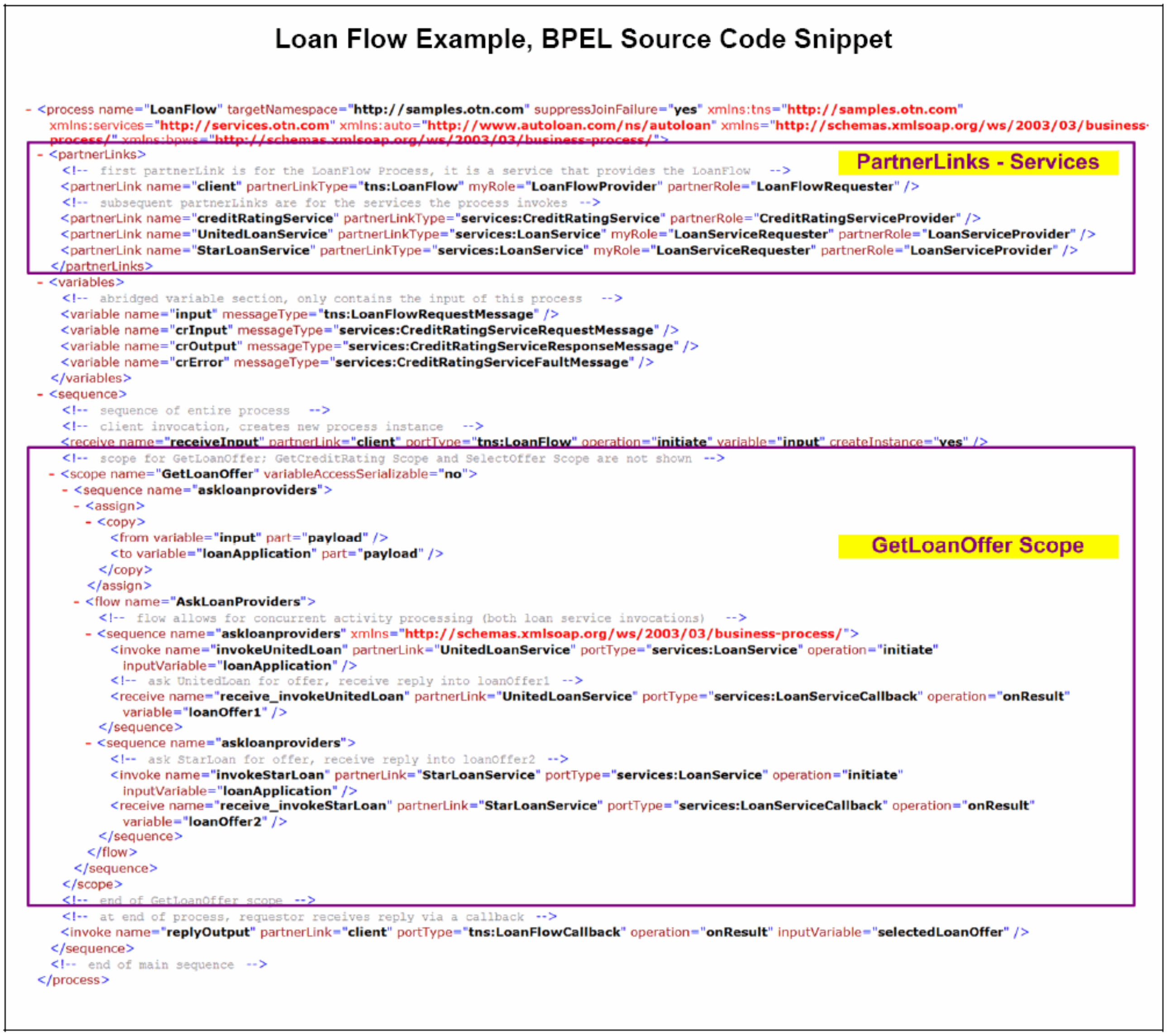
This diagram shows the Loan Flow example from a BPEL designer's view. The diagram is an an-notated composite of screen shot clips from Oracle's BPEL PM tool.

The source code view of this process is shown next. Think of this as the toggle from the design view. The code is a snippet of the actual process, focusing on the partner links and GetLoanOffer scope. In the partner links, you'll notice the first entry is for our BPEL process, with a myRole of "LoanFlowProvider," and partnerRole of requestor. The called services partner links follow; note the roles are switched. The myRole is provider, and partnerRole is requestor.

For each partner link, the partnerLinkType element points to the specific operation to be invoked, as described by the WSDL (portType) of the individual service.

Looking at the GetLoanOffer scope, you can see nested structured activities. There is an outer <sequence> to assign the request data prior to making the invocations, and an inner <flow> that allows the invocations to be concurrent.

The final activity in the process is to provide the requester with the loan offer information.



Loan Flow Example, BPEL Source Code Snippet

```
- <process name="LoanFlow" targetNamespace="http://samples.otn.com" suppressJoinFailure="yes" xmlns:tns="http://samples.otn.com"
    xmlns:services="http://services.otn.com" xmlns:auto="http://www.autoloan.com/ns/autoloan" xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-
    process/" xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
  - <partnerLinks>
      <!-- first partnerLink is for the LoanFlow Process, it is a service that provides the LoanFlow -->
      <partnerLink name="client" partnerLinkType="tns:LoanFlow" myRole="LoanFlowProvider" partnerRole="LoanFlowRequester" />
      <!-- subsequent partnerLinks are for the services the process invokes -->
      <partnerLink name="creditRatingService" partnerLinkType="services:CreditRatingService" partnerRole="CreditRatingServiceProvider" />
      <partnerLink name="UnitedLoanService" partnerLinkType="services:LoanService" myRole="LoanServiceRequester" partnerRole="LoanServiceProvider" />
      <partnerLink name="StarLoanService" partnerLinkType="services:LoanService" myRole="LoanServiceRequester" partnerRole="LoanServiceProvider" />
    </partnerLinks>
  - <variables>
      <!-- abridged variable section, only contains the input of this process -->
      <variable name="input" messageType="tns:LoanFlowRequestMessage" />
      <variable name="crInput" messageType="services:CreditRatingServiceRequestMessage" />
      <variable name="crOutput" messageType="services:CreditRatingServiceResponseMessage" />
      <variable name="crError" messageType="services:CreditRatingServiceFaultMessage" />
    </variables>
  - <sequence>
      <!-- sequence of entire process -->
      <!-- client invocation, creates new process instance -->
      <receive name="receiveInput" partnerLink="client" portType="tns:LoanFlow" operation="initiate" variable="input" createInstance="yes" />
      <!-- scope for GetLoanOffer; GetCreditRating Scope and SelectOffer Scope are not shown -->
    - <scope name="GetLoanOffer" variableAccessSerializable="no">
      - <sequence name="askloanproviders">
        - <assign>
          - <copy>
              <from variable="input" part="payload" />
              <to variable="loanApplication" part="payload" />
            </copy>
          </assign>
        - <flow name="AskLoanProviders">
            <!-- flow allows for concurrent activity processing (both loan service invocations) -->
          - <sequence name="askloanproviders" xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
              <invoke name="invokeUnitedLoan" partnerLink="UnitedLoanService" portType="services:LoanService" operation="initiate"
                inputVariable="loanApplication" />
              <!-- ask UnitedLoan for offer, receive reply into loanOffer1 -->
              <receive name="receive_invokeUnitedLoan" partnerLink="UnitedLoanService" portType="services:LoanServiceCallback" operation="onResult"
                variable="loanOffer1" />
            </sequence>
          - <sequence name="askloanproviders">
              <!-- ask StarLoan for offer, receive reply into loanOffer2 -->
              <invoke name="invokeStarLoan" partnerLink="StarLoanService" portType="services:LoanService" operation="initiate"
                inputVariable="loanApplication" />
              <receive name="receive_invokeStarLoan" partnerLink="StarLoanService" portType="services:LoanServiceCallback" operation="onResult"
                variable="loanOffer2" />
            </sequence>
          </flow>
        </sequence>
      </scope>
      <!-- end of GetLoanOffer scope -->
      <!-- at end of process, requestor receives reply via a callback -->
      <invoke name="replyOutput" partnerLink="client" portType="tns:LoanFlowCallback" operation="onResult" inputVariable="selectedLoanOffer" />
    </sequence>
    <!-- end of main sequence -->
  </process>
```

Web Service Interfaces

There are many cases where you might want to specify a web service interface. Multiple web services could then implement the interface ensuring consistency. This design pattern is generally used in federated services. For example, one might create a Professional License interface which would specify the web methods and their details for determining qualifications. This interface could then be implemented by Dentist Licensing, CPA Licensing, and Real Estate Licensing services.

An interface is used when you want to standardize a particular piece of functionality, and then apply that functionality to different scenarios.

Web Service Standards

Web services are still evolving and as a result there are a large number of standards. It is likely that some of the standards will be combined. But for now, here is a quick list of some of the more important ones:

Standards Organizations:

W3C - World Wide Web Consortium http://www.w3.org/

HTTP, CSS, SOAP, XML, XPath, XSL, WSDL, WS-Addressing, WSCI, WS Choreography Model, plus others.

OASIS – Organization for the Advancement of Structured Information Standards http://www.oasis-open.org/home/index.php

WS-BPEL, ADVL, CAP, DSML, ebXML, XACML, SAML, SPML, UDDI, UBL, WS-Reliability, WSRP,
WS-Security, WSDM plus others.

WS-I – Web Services Interoperability Organization - Provides interoperability standards in the form of Profiles. http://www.ws-i.org/ Current profiles include:

- Basic Profile (V1.0, V1.1, Simple SOAP Binding Profile 1.0)
- Attachments Profile 1.0
- Basic Security Profile (V1.0, Security Scenarios)

Liberty Alliance - http://www.cio.ca.gov/ITCouncil/Committees/ArchStandards.html

Standards:

SOAP – Originally: Simple Object Application Protocol. Now, informally referred to a SOA Protocol.
XML – eXtensible Markup Language
WSDL – Web Services Description Language
UDDI – Universal Description Discovery Integration
WSIL – Web Services Inspection Language, may eventually replace UDDI.
WS-Reliability & WS-ReliableMessaging
WSRP – Web Services for Remote Portlets
AJAX – Asynchronous JavaScript and XML

Process Standards:

BPEL – Business Process Execution Language (Microsoft, IBM). Note, now OASIS standard.
WSCL – Web Services Conversation Language (HP)
WSCI – Web Services Choreography Interface (BEA, Intalio, SAP, Sun)
BPML – Business Process Modeling Language (W3C)
BPSS – Business Process Specification Schema (ebXML)
WSFL – Web Services Flow Language (IBM)
XLANG - (Microsoft)

Transaction Standards:

WS-Transaction (WS-BusinessActivity and WS-AtomicTransaction).
WS-Coordination

Security Standards:

WS-Security
WS-Trust
WS-Provisioning
WS-Federation
WS-Authorization
WS-Policy
WS-Privacy
SAML (Secure Access Markup Language)
STS (Secure Token Service)